Combining Aspect Mining Techniques Based on Crosscutting Concern Sorts

Esteban S. Abait, Claudia A. Marcos
ISISTAN Research Institute
UNCPBA
Tandil, Argentina
{eabait, cmarcos}@exa.unicen.edu.ar

Abstract—Aspect mining encompasses techniques for identifying crosscutting concern code in legacy systems. Those techniques share several problems, including low recall and precision. In this paper, a dynamic aspect mining technique based on crosscutting concern sorts is introduced, and combined with four previous static techniques in two case studies. By combining our dynamic based technique with the other static ones we were able to improve the percentage of identified candidates 72% and 178% on average for each analyzed application. Based on the results, we reckon that static and dynamic based approaches yield complementary results and hence should be used in conjunction.

Aspect-Oriented Software Development; reengineering; aspect mining; concern sorts; dynamic analysis

I. INTRODUCTION

Evolving a legacy object-oriented software system into an aspect-oriented one is far from being a trivial task. Due to the large size of the implementation and the lack of sound documentation there is a need for tools that automate the identification, quantification and refactoring of crosscutting concerns into aspects of the new systems [8].

Aspect mining encompasses techniques that identify crosscutting concerns by reasoning about the source code or the runtime behavior of an application. These techniques are the first step in order to evolve an object-oriented version of the system into an aspect-oriented one [8]. One of the main problems with aspect mining techniques is the level of user involvement required in order to analyze the set of aspect candidates yielded by the techniques. For instance, fan-in analysis [16] returned 114 candidates for a 20KLOC system, from which only 58 candidates corresponds to crosscutting concern instances. In consequence, the precision of fan-in analysis is 51% (51 / 114). As stated by Mens et al. [15], low precision becomes a problem if the technique tends to return a lot of candidates, which decreases its scalability and ease-of-use. Another disadvantage of aspect mining techniques is low recall, i.e. the percentage of aspect candidates that corresponds to crosscutting concerns with respect to the total of crosscutting concerns in the source code.

On the other hand, Marin et al. [10] proposed a framework for combining different aspect mining techniques improving the quality of their results. The aim of such framework is ensuring consistency between the results of aspect mining techniques that address the same crosscutting concern sort. Crosscutting concern sorts are categories of atomic crosscutting concerns, which share the same intent and implementation idiom in a non-aspect-oriented language. By retrofitting three aspect mining techniques in the framework, the authors were able to combine their results, and hence improve not only the precision but also the recall values.

In this context, we argue that the precision and, particularly, the recall values of aspect mining can be improved if the combined techniques are based on different kinds of program analysis, such as static and dynamic analysis. Static and dynamic analysis are regarded as complementary techniques, static analysis is conservative and sound while dynamic analysis is efficient and precise [4]. In order to evaluate if aspect mining techniques based on dynamic analysis are complementary to static-based techniques, this paper presents the adaptation of a dynamic-based technique to the concern sorts framework, its application to two different study cases, and its combination with other static-based techniques.

Our main contributions are the followings. Firstly, the adaptation of a previous developed aspect mining technique based on dynamic analysis [1, 14] for concern sorts identification. Secondly, the presentation of the results of applying five aspect mining techniques to two cases studies: Fan-in Analysis [16], Redirection Finder and Grouped Calls [10], Aspectizable Interfaces [Cecatto and Tonella 2005], and Dynamic Analysis with Association Rules [1, 14]. Lastly, the analysis of the results of the case study to evaluate whether the hypothesis is valid. The remainder of this paper is organized as follows. Section 2 introduces the concept of crosscutting concern sorts, and the steps to adapt an existing aspect mining technique to this framework. Section 3 shows how a previous developed aspect mining technique is adapted to identify crosscutting concern sorts. Section 4 discusses the results of applying five aspect mining techniques to two study cases. Finally, Section 5 presents related work, and, in Section 6 the conclusion and future work are drawn.
TABLE I. CROSSCUTTING CONCERN SORTS

<table>
<thead>
<tr>
<th>Sort</th>
<th>Intent</th>
<th>Idiom</th>
<th>Examples (Instances)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistent Behavior</td>
<td>Implement consistent behavior as a controlled step in the execution of a number of methods that can be captured by a natural pointcut</td>
<td>Method calls to the desired functionality</td>
<td>Logging, event notification, authorization checks</td>
</tr>
<tr>
<td>Redirection Layer</td>
<td>Define an interfacing layer to an object and forwards the calls to the object</td>
<td>Declare a routing layer and have methods in this layer to forward the calls</td>
<td>Decorator and Adapter Patterns [Gamma et al. 1994]</td>
</tr>
<tr>
<td>Role Superimposition</td>
<td>Implement a specific secondary role or responsibility</td>
<td>Interface implementation, or direct implementation of methods that could be abstracted into an interface definition</td>
<td>Roles specific to design patterns like Subject for an Observer pattern; Persistence</td>
</tr>
</tbody>
</table>

II. CROSSCUTTING CONCERN SORTS

Crosscutting concern sorts are atomic descriptions of crosscutting functionality. They are characterized by a number of properties common to all the instances of the sort, such as a generic description of the sort (its intent), and a specific implementation idiom of the sort's instances in a non-aspect-oriented language (i.e., the sort's specific symptom) [10]. For instance, typical implementations of tracing [9], authorization checks [9], or the notification mechanism in the Observer pattern [5] follow the same idiom that consists of method invocations. Those methods provide common functionality required from, or imposed on the participants in a given context and are scattered through the rest of the implementation [3]. In this case, the implementation idiom is regarded as a consistent behavior sort. Table 1 shows three crosscutting concern sorts [10]. For each sort, its intent and idiom are described and examples are presented.

In order to fit an aspect mining technique to identify crosscutting concern sorts, the following issues must be addressed [10].

1. **Search-goal** of the mining technique (which sort will identify).
2. **Representation** of the mining results. This is the format for presenting the results of the automatic mining process (e.g., call relations or concepts in a lattice).
3. **Mapping** between the mining results and the search-goal.
4. **Assessment metrics** to evaluate mining techniques and its results.

The classification of crosscutting concerns based on sorts ensures a number of important properties for consistent aspect mining [10]. First, the atomicity of the sorts ensures a consistent granularity level for the mining results. Second, sorts describe the relation between concrete instances and associated crosscutting functionality. Third, sorts provide a common language for referring to typical crosscutting concerns code, and, hence, for defining the search-goals of an aspect mining technique.

III. ADAPTING A PREVIOUS APPROACH

This section describes the adaptation of the approach presented in [1, 14] into a sort-oriented aspect mining technique. Our previously proposed approach employs association rule algorithms [2] to analyze execution traces and find potential aspect candidates. The result of this adaptation is a technique that identifies instances of consistent behavior, redirection layer and role superimposition sorts.

Our previous approach comprises three main phases. The first phase consists in the collection of execution traces and the call-graph through system execution according to a set of execution scenarios. An execution trace consists of methods invoked during a system execution, whereas the call-graph registers the calling relations within methods invoked during runtime. The second phase takes as input the execution traces and applies association rules algorithms to them. The third phase post-process association rules using a set of post-processing filters. Those filters discard some rules and identify others as aspect candidates. The call-graph is used by the filters so as to know whether a method invoked another one in a system execution. Finally, the resulting association rules are the aspect candidates.

The adaptation of the approach was made in the second and third phases. Three filters were developed that target three kinds of crosscutting concern sorts. The previous approach filtered association rules after they were generated, by contrast, this approach is able to identify aspect candidates during the generation of frequent itemsets [2] and association rules. As a result, the overall approach is more efficient.

Frequent itemsets are a kind of data mining pattern related to co-occurrence of items in a database transaction [6]. In this context, a frequent itemset relates methods that co-occur in execution traces, as often as indicated by a support metric. The support of an itemset \([ml, m2]\) indicates the percentage of execution traces that includes the methods \(ml\) and \(m2\). An association rule, on the other hand, is a rule of the form “if-\(ml\)-then-\(m2\)” which relates the occurrence of the method \(m2\) to the occurrence of the
TABLE II. RESULTS FOR JHotDraw

<table>
<thead>
<tr>
<th>Technique</th>
<th>No. Cand</th>
<th>Abs. Recall</th>
<th>Precision</th>
<th>Imp. Precision</th>
<th>Imp. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan-in (FI)</td>
<td>109</td>
<td>33</td>
<td>30%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Red. Finder (RF)</td>
<td>13</td>
<td>12</td>
<td>92%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Grouped Calls (GC)</td>
<td>22</td>
<td>12</td>
<td>55%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Asp. Int. (AI)</td>
<td>7</td>
<td>7</td>
<td>100%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Role Filter (RoleF)</td>
<td>34</td>
<td>18</td>
<td>53%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CB Filter (CBF)</td>
<td>6</td>
<td>2</td>
<td>33%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Red. Layer Filter (RLF)</td>
<td>18</td>
<td>10</td>
<td>56%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RoleF ∩ AI</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>RoleF U AI</td>
<td>39</td>
<td>23</td>
<td>59%</td>
<td>-41%</td>
<td>229%</td>
</tr>
<tr>
<td>RF ∩ RLF</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td>9%</td>
<td>-83%</td>
</tr>
<tr>
<td>RF U RLF</td>
<td>29</td>
<td>20</td>
<td>69%</td>
<td>-123%</td>
<td>67%</td>
</tr>
<tr>
<td>FI ∩ CBF</td>
<td>5</td>
<td>2</td>
<td>40%</td>
<td>33%</td>
<td>-94%</td>
</tr>
<tr>
<td>FI U CBF</td>
<td>110</td>
<td>33</td>
<td>30%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>GC ∩ CBF</td>
<td>3</td>
<td>2</td>
<td>67%</td>
<td>21%</td>
<td>-83%</td>
</tr>
<tr>
<td>GC U CBF</td>
<td>25</td>
<td>12</td>
<td>48%</td>
<td>-13%</td>
<td>0%</td>
</tr>
</tbody>
</table>

method $m_1$ according to the value of the confidence metric. The confidence metric denotes the conditional probability of $m_1$ given $m_2$.

The third phase was modified in order to group frequent itemsets or association rules that belong to the same seed. Consequently, aspect candidate are group of one or more itemsets or association rules.

Following, the filters integrated within the mining algorithms are presented. Each filter is described alongside its search goal, presentation format and mapping. The three filters are assessed with the same metrics: precision and absolute recall. Precision reports the percentage of seeds that correspond to a crosscutting concern from all the seeds reported. The absolute recall metric measures the total number of seeds that pertain to a concern sort.

A. Redirection Layer Sort Filter

Redirection layer sort filter tries to identify redirector methods on frequent itemsets. A redirector method redirects their callers to dedicated methods in another class.

This filter looks for itemsets of size two, where the methods contained in the itemset share the signature and, one of the methods invokes the other one during execution. The identified itemsets are grouped according to the classes of both methods.

- **Search goal**: Instances of redirection layer sort.
- **Presentation**: Group of itemsets that share the same redirector and dedicated classes: [$A.m_1, B.m_1; A.m_2, B.m_2$].
- **Mapping**: One of the classes in the itemsets matches the redirector type and the other one match the dedicated class.

B. Role Superimposition Sort Filter

Role superimposition sort filter looks for classes that fulfill multiple roles or responsibilities. To detect seeds of this sort, the filter looks for itemsets containing methods that share the same name. The support value ($v$) associated with each itemset ensures that the methods contained in the itemset take part on the realization of at least $s$ execution scenarios. In consequence, both methods were called during the system execution for more than one functional scenario, thus both methods may be implementing crosscutting behavior. The identified itemsets are grouped by method name.

- **Search goal**: Instances of role superimposition sort.
- **Presentation**: Group of itemsets that share the same name: [$A.m_1, C.m_1; D.m_3, E.m_3$].
- **Mapping**: The classes of the methods in the itemsets implement secondary roles or responsibilities.

C. Consistent Behavior Sort Filter

Consistent behavior sort filter aims at identifying methods which provide a common action and are used through the rest of the implementation. This filter looks for association rules that have the same method included in the consequent but different antecedent. The identified rules are grouped according to the consequent they share.

- **Search goal**: Instances of consistent behavior sort.
- **Presentation**: Group of association rules that share the same consequent: [$A \Rightarrow B; C \Rightarrow B; D \Rightarrow B$].
- **Mapping**: The method included in the consequent provides the crosscutting behavior. The methods included in the antecedent represent the context in which the common behavior is required.
IV. Case Study

In this section, two case studies are presented and the results of applying the proposed aspect mining technique alongside with other four static-based techniques are shown.

The static techniques were chosen in order to target the same crosscutting concern sorts as the proposed technique: aspectizable interfaces [Tonella and Ceccato 2004b] target role superimposition sorts, fan-in and grouped calls [10] target consistent behavior sorts, and redirector finder [10] target instances of redirection layer sorts.

Two applications were considered for this case study: JHotDraw 5.4b1 [7] and TinyUML 0.13.02 [13]. JHotDraw is a Java object-oriented framework, with approximately 18,000 non-commented lines of code and around 2800 methods. JHotDraw is a framework for drawing structured 2D graphics and was originally developed as an exercise to illustrate good use of object-oriented design patterns [Gamma et al. 1994]. TinyUML, on the other hand, is an UML 2.0 editor with approximately 8,000 non-commented lines of code.

Tables 2 and 3 present the results for both applications.

The 'No. Cand' column indicates how many aspect candidates were generated for each technique, or combination of them. The 'Abs. Recall' column shows the number of aspect candidates that belong to a concern sort. The 'Precision' column indicates the percentage of candidates that correspond to concern sorts. The 'Imp. Precision' and 'Imp. Recall' shows the increase in precision and absolute recall when combining the results of the dynamic and static techniques in comparison to the static technique alone. For instance, when calculating the union of AI results with RoleF results, the precision shows a 41% drop down compared to the precision of AI alone, and the absolute recall shows an increase of 229% compared to the same metric for AI.

For JHotDraw, on average, the precision increased 16\% (RoleF \( \cap \) AI + RF \( \cap \) RLFI + FI \( \cap \) CBF + GC \( \cap \) CBF) / 4, and the absolute recall rised up to 72\%, (RoleF \( \cup \) AI + RF U RLFI + FI U CBF + GC U CBF) / 4. The Role Filter presented the greatest increase in absolute recall, since this filter was able to identify instances missed by the AI technique. The AI statically looks for classes that implement interfaces whose names include the string '*able', like 'Storable'. The Role Filter, by contrast, was able to identify interfaces like 'ViewChangeListener' or 'MouseMotionListener', improving the final absolute recall. The CB Filter showed a 0\% improvement in absolute recall since its results are subsumed by the results of FI and GC, as a result the use of dynamic techniques showed no benefit regarding the identification of this sort. The increase in the precision, on the other hand, is rather low compared to the absolute recall increase.

Concerning TinyUML, on average, the precision shows a 100\% drop down, although the absolute recall goes up dramatically to 178\%. Again, the improvement in absolute recall exceeds the improvement in precision, which in fact shows a decrease. This decrease can be explained by the fact that for the following combinations: RoleF \( \cap \) AI, RF \( \cap \) RLFI, FI \( \cap \) CBF and GC \( \cap \) CBF the techniques results were complementary yielding a 0\% precision and hence a 100\% decrease when compared with the precision of the static approach. What is more, this complementary in the generated candidates yielded a rise in the absolute recall percentages of 140\%, 400\% and 150\% respectively.

Marin et al. [10] stated that in order to improve absolute recall the union of the results of techniques addressing different crosscutting concern sorts should be considered. However, the previous results showed that the union of the results of techniques addressing the same concern sort, but based on different analysis, can better the absolute recall.

Moreover, Marin et al. suggested that intersecting the results of techniques that look for the same crosscutting concern sort will increase the precision. Nonetheless, when intersecting results in the TinyUML experiment, the precision showed a drop down due to the fact that the intersection of the results was empty.

\footnote{All results and tool outputs are available at: http://sites.google.com/site/legacyandaop/Home/am}
On balance, this result empirically proved that the use of static and dynamic techniques is complementary and should be applied in order to enhance the number of detected aspect candidates.

V. RELATED WORK

Previous work on crosscutting concern sorts was carried out by Marin et al. in [10, 11, 12]. These works include the previously described framework for combining and assessing aspect mining techniques, a way to document concern sorts in source code, and a systematic process for refactoring of object-oriented systems towards aspects. This work provides further assessment on the use of the framework. In particular, we report the results of combining static and dynamic aspect mining techniques in two case studies and an evaluation of the combined precision and absolute recall. A previous experience combining aspect mining approaches [3] also showed that dynamic-based techniques were able to discover aspect candidates missed by the static-based techniques. However, the authors conducted a qualitative analysis and do not provide concrete measures on the combined precision or absolute recall.

VI. CONCLUSIONS AND FUTURE WORK

This paper shows the adaptation of a previously aspect mining approach based on dynamic analysis to fit into the crosscutting concern sorts framework proposed by Marin et al. [10]. In addition, the adapted technique was applied in two applications in conjunction with other four static-based aspect mining techniques. The results of this case study, suggest that both dynamic-based and static-based aspect mining techniques yields complementary results, improving the number of discovered concern and, indirectly, the total recall. For future work, firstly, we want to analyze more and larger systems, including distributed ones. Secondly, we will compare the results of our technique with other proposed techniques based on dynamic analysis. Additionally, we want to analyze the impact that the support and confidence values have on the results generated by the proposed technique.

VII. REFERENCES


